Design Methodology for VLSI Implementation of Image and Video Coding Algorithms – A Case Study

Javier Bracamonte
Michael Ansorge
Fausto Pellandini

ABSTRACT In this chapter a methodology for the design of VLSI circuits for image and video coding applications is presented. In each section a different phase of the design procedure is discussed, along with a description of the involved software environments. An example of an area efficient single-chip implementation of a JPEG coder is presented to illustrate the methodology.

1 Introduction

To produce an image in digital form, an enormous amount of data is required. Due to the explosive proliferation of digital images in today's world, the prodigality of digital image data has three direct economic and technical implications: a) the storage of digital images demands a significant amount of memory media, b) the transmission of digital images requires long transmission times, c) image processing applications need a large computational power.

In their original form (just after they have been digitized), two images with the same spatial resolution require exactly the same number of data (bits) regardless of their information content. For example, an image of a cloudless sky would require the same number of bits as an image of a crowd in a stadium, even though the latter contains much more information than the former. Technically speaking, it is said that the former image contains a high degree of redundancy, in the sense that, in exchange for some processing, the same image could be equally described with less data.

The kind of redundancy pointed out in the previous paragraph is only one among others [1]. Image coding—or compression—is the art or science of eliminating all possible kinds of redundancies in a digital image, so as to represent it with the least possible data, while retaining most of the original information. The same definition is valid for video coding, where another and exploitable kind of redundancy is present, given the high degree of similarity between two successive images in a video sequence.

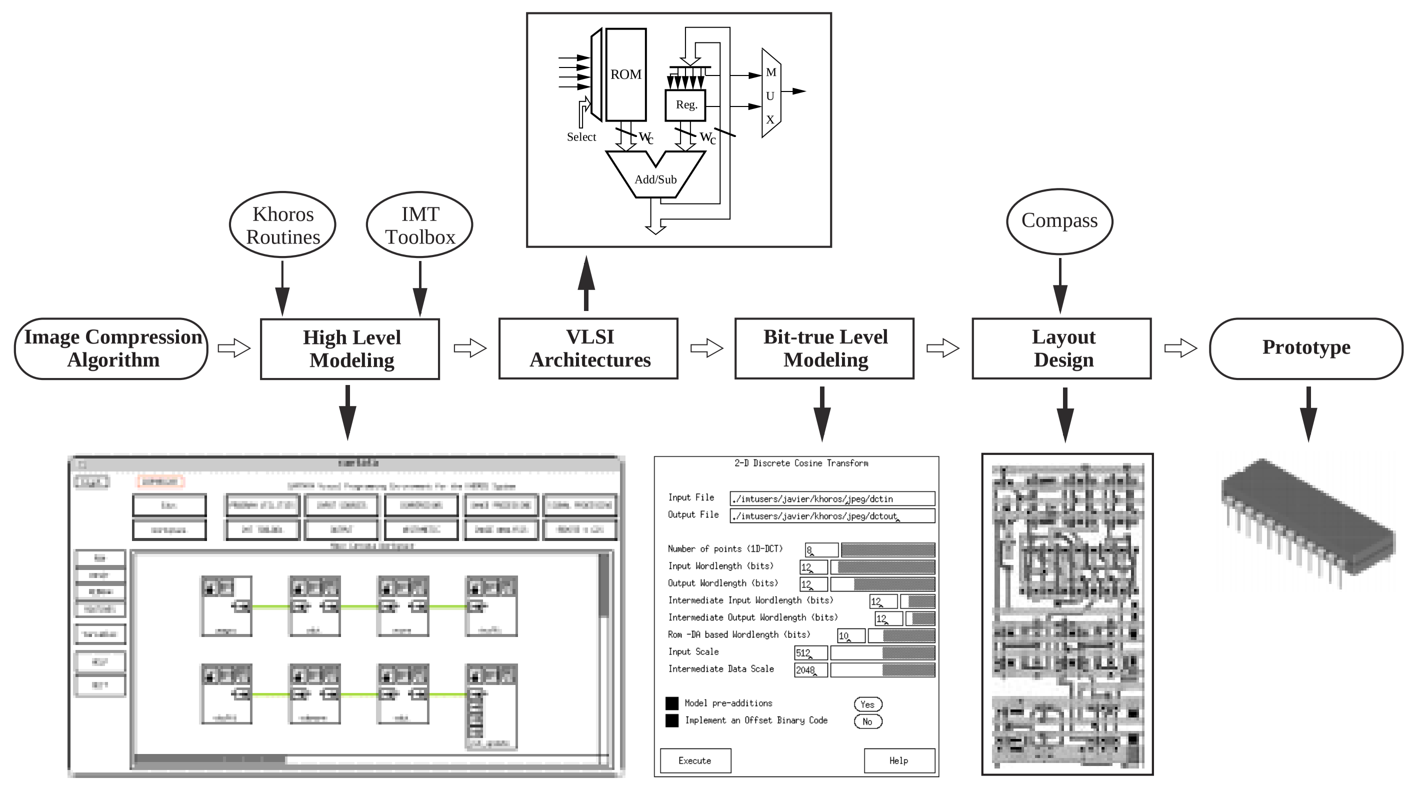
Several image compression techniques are currently available [2][3]. In general, the original image is processed with the selected coding algorithm, to produce a compressed version. Depending on the application, the compressed version is either stored, or transmitted via a communication channel. Then, when the original image is required, the decoding algorithm is applied to the compressed version to recover an exact copy of the original image (lossless compression), or a very good replica (lossy compression). In effect, to increase the compression efficiency, most of the coding algorithms are allowed to introduce some errors.

In order to produce a compressed (or decompressed) image or video sequence, a significant amount of computational resources are needed. For some applications, a general purpose personal computer would suffice to execute the coding/decoding algorithms. However, when the process must be performed in real time, the required computational power might be so high that it can only be achieved by means of particular VLSI circuits. These circuits may correspond to special programmable video signal processors (VSP) of the last generation [4][5], or to application specific integrated circuits (ASICs). The ASIC solution, towards which the methodology presented in this chapter is oriented, is essential for particularly demanding applications such as miniaturized and/or portable equipment.

The process of materializing the mathematical description of an image compression algorithm into a chip implies the solution of a relatively complex problem. The complete design process extends over many levels of abstraction; each level requiring the use of different methods and technologies, whose intermediate results should be useful to facilitate the tasks of the next abstraction level. An effective methodology is thus indispensable to orchestrate the multiple involved procedures in order to reduce time, effort and design errors, while constraining the final result for a particular feature such as high speed, small area or low power consumption.

The different steps of the methodology reported in this chapter are: a) high-level modeling of the algorithm, b) VLSI architecture design, c) bit-true level modeling, d) layout design, and e) verification and testing. Figure 1 shows the process stream of the methodology, along with the supporting software environments. To illustrate the methodology, a case study consisting of the VLSI implementation of a JPEG coder will be described.

The remainder of this chapter is organized as follows. A brief description of the JPEG algorithm is presented in section 2. The high level modeling is described in section 3. The VLSI architectures are presented in section 4. The bit-true level modeling of the algorithm is described in section 5, and the layout design is discussed in section 6. Finally, the results are given in section 7, just before the conclusions in section 8.



2-D Discrete Cosine Transform

Input File ./imtusers/javier/khoros/jpeg/dctin
Output File ./imtusers/javier/khoros/jpeg/dctout

Number of points (1D-DCT) 8
Input Wordlength (bits) 12
Output Wordlength (bits) 12
Intermediate Input Wordlength (bits) 12
Intermediate Output Wordlength (bits) 12
Rom -DA based Wordlength (bits) 10
Input Scale 512
Intermediate Data Scale 2048

Model pre-additions
Implement an Offset Binary Code

FIGURE 1. Methodology.

2 JPEG Baseline Algorithm

The Joint Photographic Expert Group (JPEG) algorithm defines an international standard for compression of continuous-tone still images. It specifies four modes of operation: sequential, progressive, lossless and hierarchical [6]. Thus, rather than a single image compression method, JPEG defines a family of application-dependent coding schemes. Among them, the baseline system of the sequential mode is by far the most used, since it covers a wide range of applications. It is hence the implementation system we address in this chapter.

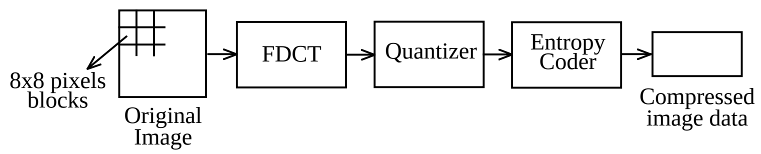


FIGURE 2. Baseline JPEG algorithm.

The baseline sequential JPEG algorithm is depicted in Figure 2. It is a Discrete Cosine Transform (DCT) -based process which transforms blocks of 8x8 pixels sequentially from the original image, into 8x8 blocks of coefficients in the frequency domain [7]. The goal of this transformation is to decorrelate the original data and redistribute the signal energy among only a small set of transform coefficients in the low frequency zone. For illustration, an array of 8x8 pixels, corresponding to a region of the eye in an image of a person's face, is given in Equation (1.1). The result after a 2-D DCT has been applied to the array **x** (whose elements are previously offset by -128, as required by JPEG) is given in Equation (1.2). It can be seen that most of the energy is concentrated in only a few low frequency (i.e., the top left corner region) coefficients.

$$\mathbf{x} = \begin{bmatrix} 77 & 69 & 69 & 70 & 71 & 67 & 67 & 67 \\ 74 & 70 & 72 & 67 & 64 & 66 & 66 & 65 \\ 67 & 72 & 68 & 68 & 64 & 66 & 68 & 66 \\ 65 & 72 & 71 & 71 & 62 & 65 & 69 & 90 \\ 68 & 76 & 79 & 65 & 61 & 56 & 58 & 99 \\ 78 & 93 & 95 & 69 & 59 & 56 & 56 & 71 \\ 73 & 108 & 106 & 92 & 68 & 60 & 60 & 83 \\ 70 & 105 & 110 & 105 & 90 & 84 & 83 & 102 \end{bmatrix} \tag{1.1}$$

$$\mathbf{X} = \begin{bmatrix} -431 & 25 & 11 & -48 & -3 & -16 & 4 & -6 \\ -52 & -16 & 6 & 40 & 8 & 15 & 0 & 5 \\ 35 & 9 & -20 & 4 & -5 & 4 & -6 & 0 \\ -18 & 19 & 11 & -2 & 2 & -3 & 6 & 6 \\ 15 & -26 & 2 & -3 & 11 & -3 & 2 & -1 \\ -1 & 6 & -3 & -2 & -1 & 1 & -5 & -4 \\ -1 & 1 & -4 & 9 & -4 & 2 & -1 & 3 \\ -2 & -1 & 0 & -4 & 4 & -4 & -1 & -1 \end{bmatrix} \tag{1.2}$$

Based on a psychovisual analysis, a normalization array can be defined. Its purpose is to quantize those DCT coefficients that are visually significant with relative short quantization steps, while using large quantization steps for those coefficients which are less important. This large-step quantization, associated with the energy packing effect of the transformation, results in general in the zeroing out of many DCT coefficients. Equation (1.3) gives the resulting array $\mathbf{Q}$, when the matrix $\mathbf{X}$ in Equation (1.2) has been quantized by using the normalization matrix given in Equation (1.4) on page 8.

$$\mathbf{Q} = \begin{bmatrix} -27 & 2 & 1 & -3 & 0 & 0 & 0 & 0 \\ -4 & -1 & 0 & 2 & 0 & 0 & 0 & 0 \\ 2 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{1.3}$$

A long sequence of zero-valued coefficients can be effectively abridged by runlength coding. In order to increase the efficiency of the runlength coding, the array of normalized DCT coefficients is reordered by following a zigzag pattern. The reordering has the effect of producing longer runs of zeros. The resulting zigzag-reordered, 64-element 1-D array is: $\{-27, 2, -4, 2, -1, 1, -3, 0, 1, -1, 1, 1, -1, 2, 0, 0, 0, 0, 0, -1, 0, \ldots, 0\}$, where the ellipsis replaces a sequence of 42 zeros.

The top-left coefficient (element $(0, 0)$) of the array $\mathbf{X}$ in Equation (1.2), is called the DC coefficient (the remaining 63 coefficients are referred to as AC coefficients). Its value represents the (upscaled) average value of the 64 pixels of the original image (this value in Equation (1.2) is negative due to the offset of -128, as pointed out above). Since the average value of two adjacent 8x8 blocks of pixels is, with a high probability, very similar, a straight DPCM (differential pulse code modulation) scheme can be applied to the DC coefficients, using the DC coefficient of the previous coded block as the prediction value.

Though the quantization of the DCT coefficients is the main mechanism of data compression (and also of information loss), additional data compression can be obtained by entropy coding the output of the DPCM and runlength coders. While both Huffman and arithmetic coding are defined by the general JPEG standard for the entropy coding stage, the baseline JPEG algorithm uses Huffman coding only. This fact presents two important advantages: first, Huffman coding requires a less complex hardware implementation; and second, many algorithms on arithmetic coding are patented, and their commercial implementation, in consequence, requires the payment of license fees [8].

3 High Level Modeling

A modular high level implementation of the JPEG algorithm is shown in Figure 3. The top and bottom flowgraphs represent the encoder and decoder respectively. The runlength coding and DPCM operations, described in the previous section, were merged into a single module with the Huffman coder/decoder (i.e., *vhuffc*/*vhuffd*). From an image processing point of view, this high-level modeling is an application program by itself [9]. Typical compression ratios of images are around 10, depending on the image activity and spatial resolution.

In the high level model, each process of the algorithm was implemented with the C programming language, using floating point precision for the arithmetic operations. No concerns were raised at this point on issues as to how the operations will be carried out by the hardware (e.g., hardware multiplexing, parallel processing, bit-serial architectures, etc.). With respect to our methodology, developing a high level model of an algorithm allows us to have both a reference and a framework to support the bit-true level modeling (section 5).

The Khoros software [10] has been chosen as the environment for developing the high and bit-true level modeling. Due to its user-friendliness and visual programmable capability, it permits a rapid evaluation of the results of a given configuration. Each glyph (or icon) in Figure 3 corresponds to a C program that has been converted into a Khoros routine. The glyph named *images* addresses a database and provides the image to be coded. The glyph *put_update* has the task of displaying the decompressed image. Both routines are part of the original Khoros software.

The graphical user interface (GUI) of the routine *vdct* is shown in Figure 4. Though JPEG specifies a fixed block size of 8x8 pixels, the number of points of the inherent 1-D DCT has been left as a parameter for experimental purposes. The same GUI is used for both the forward and inverse 2-D DCT.

The GUI of the routine *vnorm* is shown in Figure 5. It performs the

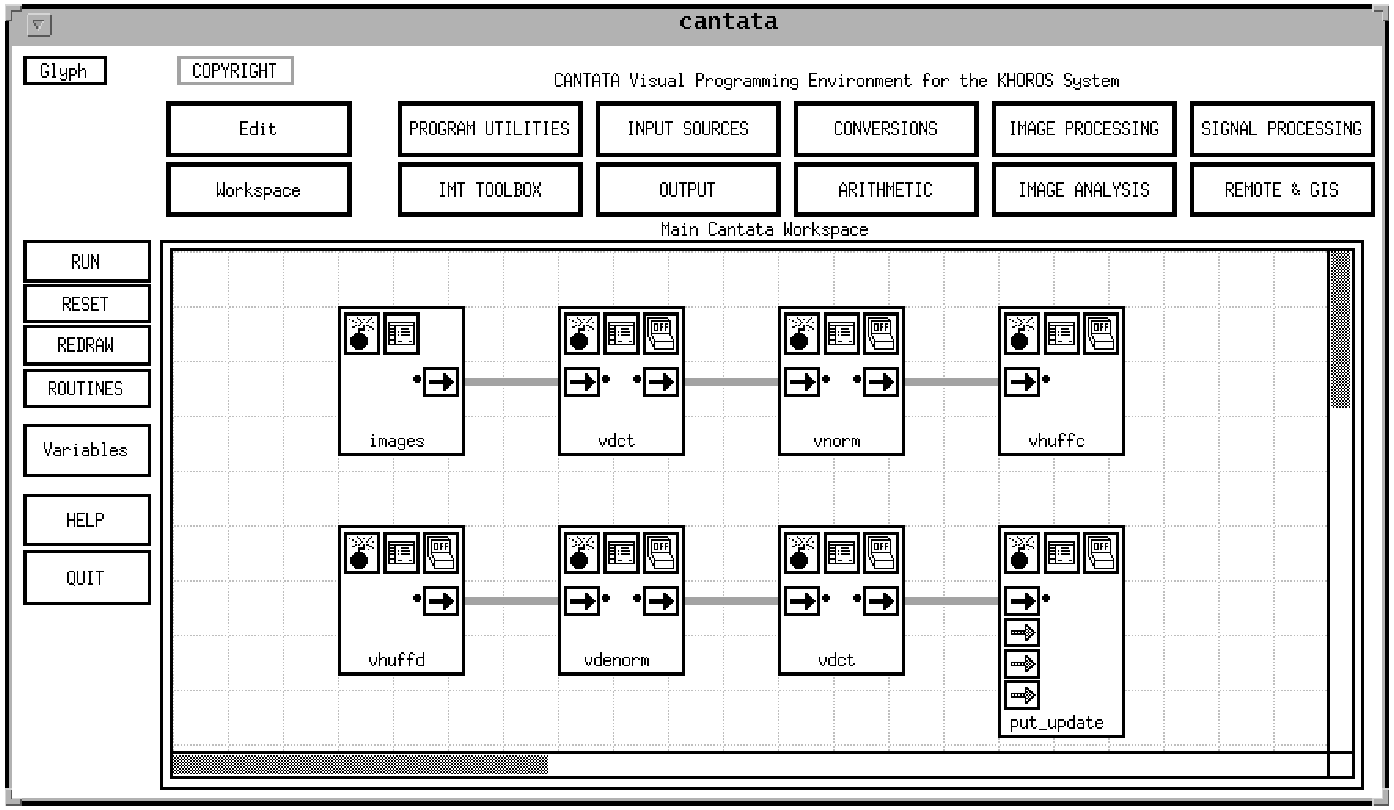
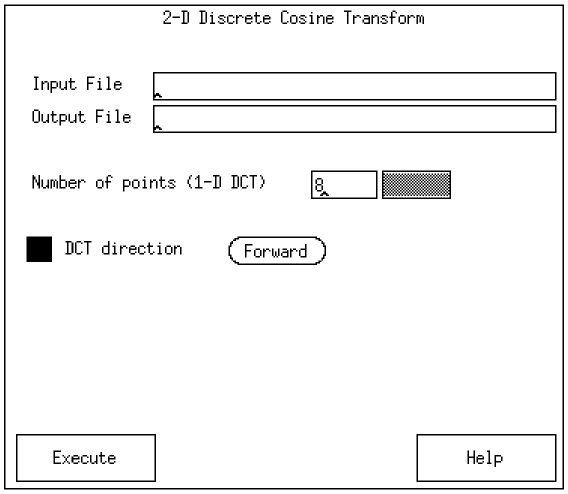


FIGURE 3. High level implementation of JPEG in Khoros.



2-D Discrete Cosine Transform

Input File

Output File

Number of points (1-D DCT) 8

DCT direction Forward

FIGURE 4. GUI of the 2-D DCT.

quantization of the 2-D DCT coefficients with a normalization array that is user selectable. By default the normalization array given in Equation (1.4) has been programmed; it corresponds to the array proposed in Table K.1 in Annex K of reference [11]. The *Scale Factor* parameter allows the user to control the compression ratio. The higher the setting of this scale factor the lower the amount of the resulting compressed image data. This kind of improvement of the compression ratio (CR) is made to the detriment of the quality of the reconstructed image. By default, *Scale Factor* is set to 1, which offers a good compromise: image quality/CR.

$$\mathbf{N} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \tag{1.4}$$

The *vhuff* routine implements the Huffman coding. The user is given the possibility of selecting a Huffman table customized to the application. By default, the Huffman codes proposed in Table K.5 in Annex K of reference [11] have been programmed.

As the number of new developed Khoros routines grows, it is convenient to integrate them into a toolbox. This provides a software structure that is easier to reuse, maintain, document, and distribute [12]. The created IMT Toolbox, contains the JPEG related routines shown in Figure 3, and some routines for other methods for image compression.

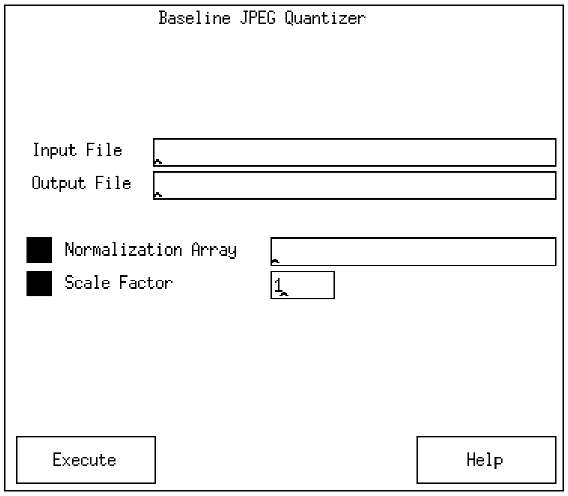


FIGURE 5. GUI of the quantizer.

4 VLSI Architectures

To introduce this section, it is worth making a point regarding the design flowgraph in Figure 1. Although it is not explicitly shown in Figure 1, there is a bidirectional interaction among the different processes. When a proposed scheme turns out not to be a good solution, the designer should go back and forth in the design process while exploring other approaches. The previous remark is particularly pertinent to the design of the VLSI architectures. Thus, the reader should be aware that the solutions presented in this section are the result of several evaluations and trade-offs.

Taking into consideration the constraints of the target application, a VLSI architecture for each block in Figure 2 is to be developed. As pointed out in the introduction, given their associated high data throughput, image and video coding applications require particular hardware implementations that operate at high clocking frequencies. If the compression circuits are intended to be used in portable equipment, then the power consumption is also of paramount importance.

Even though the relentless advance in the development of new semiconductor technologies makes feasible the production of devices that run faster or consume less power, a good architecture is nevertheless the key to achieving the specifications of a given application. Low power VLSI for image processing applications is currently a hot research topic. Though it was an issue that was considered during the design of the architectures below, the main objective remained the realization of high speed and small silicon area circuits.

Bit-serial architectures present several advantages [13]: a) the size of bit-serial structures is considerably smaller than their bit-parallel counterparts; b) wiring, which in a VLSI circuit requires a large part of silicon area, is dramatically reduced, since a single wire is required for each input word; c) communication between bit-serial modules is easier to implement, and d) bit-serial architectures lead to tightly pipelined structures at the bit-level, which implies that a maximum clock-rate can be achieved [14]. Thus, whenever it was possible, we have chosen a fully pipelined bit-serial approach.

In the following paragraphs we describe the VLSI architecture for each of the main modules of the encoder in Figure 2.

4.1 FDCT

The forward 2-D DCT appropriate for JPEG is defined in [11] as:

$$X_{uv} = \frac{1}{4} C_u C_v \sum_{i=0}^{7} \sum_{j=0}^{7} x_{ij} \cos[(2i+1)u\pi/16] \cos[(2j+1)v\pi/16]; \quad (1.5)$$

for $u, v = 0, 1, \ldots, 7$, where $C_u, C_v = 1/2$ for $u, v = 0$; $C_u, C_v = 1$ otherwise. X_{uv} represents the value of the 2-D DCT coefficient at the point (u, v) in the frequency domain, and x_{ij} represents the value of the image pixel at the point (i, j). Given that the 2-D DCT is separable, it can be reformulated as two successive 1-D DCTs: the first one being applied to the rows of the block of 8x8 pixels, whereas the second is applied to the columns of the resulting matrix (or to the rows of its transpose) [7]. This decomposition leads to a simpler hardware implementation.

Using matrix notation, the N-point 1-D DCT $\mathbf{X}$, of an input vector $\mathbf{x}$, can be expressed as:

$$\mathbf{X} = \mathbf{A}\,\mathbf{x}, \quad (1.6)$$

where $\mathbf{A}$ represents the cosine transformation matrix. For $N = 8$ (as required by the JPEG standard) each element of $\mathbf{X}$ is obtained by means of an 8-point inner product of the following form:

$$X_k = \sum_{l=0}^{7} A_{kl}\, x_l, \quad \text{for } k = 0, 1, \ldots, 7. \quad (1.7)$$

Distributed arithmetic [15] is appropriate for the calculation of inner products. It has the interesting property of circumventing all the multiplications, which are elegantly replaced by the less complex addition and shift operations.

Let us consider the evaluation of the following inner product:

$$y = \mathbf{a}\,\mathbf{x} = \sum_{i=0}^{N-1} a_i\, x_i, \quad (1.8)$$

where **x** and **a** denote an input signal vector and a fixed coefficient vector, respectively.

If x_i is represented as a W_d bits binary number in 2's complement form, then it can be expressed as:

$$x_i = \sum_{j=1}^{W_d-1} x_{ij} 2^{-j} - x_{i0}, \tag{1.9}$$

where x_{ij} represents the jth bit of x_i. Substituting Equation (1.9) into (1.8) and interchanging the order of the summations, we obtain:

$$y = \sum_{j=1}^{W_d-1} \left[\sum_{i=0}^{N-1} a_i \, x_{ij} \right] 2^{-j} - \sum_{i=0}^{N-1} a_i \, x_{i0}. \tag{1.10}$$

Let us define F_j as:

$$F_j = \sum_{i=0}^{N-1} a_i \, x_{ij}, \tag{1.11}$$

since the coefficients a_i are fixed, and x_{ij} is a binary number, then any evaluation of F_j must be a number among only 2^N possible values. In a hardware implementation, these 2^N values can be simply stored in memory (e.g., a ROM).

By substituting Equation (1.11) into (1.10), the original expression of the scalar product of Equation (1.8) can be rewritten as:

$$y = \sum_{j=1}^{W_d-1} F_j 2^{-j} - F_0, \tag{1.12}$$

which can be implemented by a ROM for storing the values of F_j, an accumulator to implement the sum, and a shifter for executing the multiplications by 2^{-j}.

The architecture of the 2-D DCT is shown in Figure 6. Each 1-D DCT is executed by a single, multiplexed, distributed arithmetic processor (DAP) [16], whose architecture is shown in Figure 7. For the computation of each of the eight coefficients X_k, in Equation (1.7), the input vector **x**, is applied eight consecutive times to the DAP. Each coefficient is computed by using a different lookup table. Thus, the ROM depicted in Figure 7, contains eight different ROMs which are chosen with the signal *Select*. The shift register bank (SRB) in Figure 6 provides the storage and sequential dataflow required for the multiple applications of the input vector.

It turns out that the cosine transformation matrix in Equation (1.6) presents a symmetry on its even rows and an antisymmetry on its odd rows. This implies that Equation (1.7) can be rewritten as:

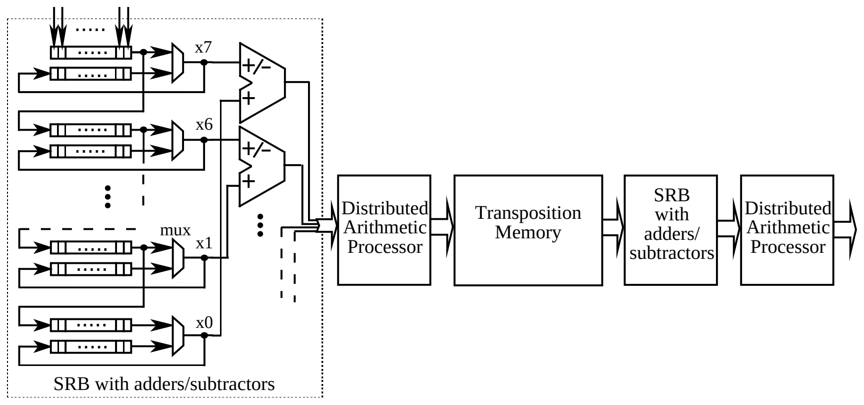

x7
+/-
+
x6
mux
x1
x0
SRB with adders/subtractors
Distributed Arithmetic Processor
Transposition Memory
SRB with adders/ subtractors
Distributed Arithmetic Processor

FIGURE 6. Architecture of the 2-D DCT.

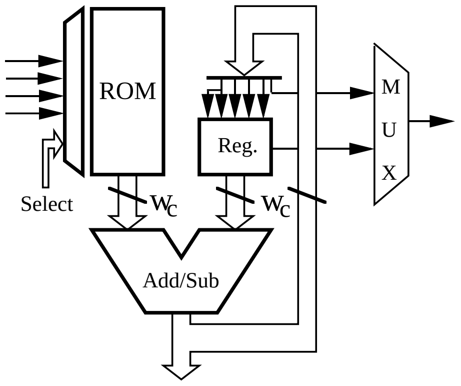


FIGURE 7. Distributed arithmetic processor.

$$X_k = \sum_{l=0}^{3} A_{kl}\,(x_l \pm x_{8-l}), \quad \text{for } k = 0, 1, \ldots, 7 \tag{1.13}$$

where the plus sign is used for the even values of k, and consequently, the minus sign for the odd values. The importance of Equation (1.13) is evident: the number of multiplications has been halved. In terms of the DAP's hardware, and in accordance with Equation (1.11), this means that the size of the DAP's ROMs in Figure 7, can been reduced from 2^N to $2^{N/2}$ words. For $N = 8$, eight small ROMs of 16 (instead of 256) words each, are required. Due to this significant memory saving, Equation (1.13) was retained for implementation. The adders/subtractors in Figure 6 perform the extra additions/subtractions in accordance with Equation (1.13).

4.2 Quantizer

The second operation of the JPEG coder is the quantization of the 2-D DCT coefficients and it is defined as: $Cq_{ij} = round(C_{ij}/N_{ij})$ for $i, j = 0, 1, ..., 7$, where C_{ij} denotes the ij-th element of an 8x8 DCT coefficients matrix, whereas N_{ij} denotes the ij-th element of an 8x8 normalization matrix. Reformulating the previous equation as $Cq_{ij} = round(C_{ij} * (1/N_{ij}))$ allows the replacement of the circuitry of a divisor with that of a simpler multiplier. The VLSI architecture of the quantizer is shown in Figure 8. Since the

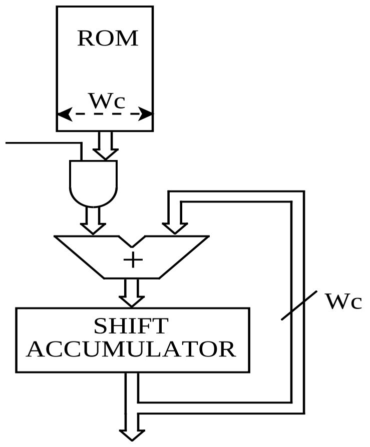


FIGURE 8. Architecture of the quantizer.

output of the 2-D DCT processor is bit-serial, a serial-parallel multiplier was implemented. The parallel input to this multiplier being the output of a ROM that contains the inverse of the normalization coefficients.

4.3 Entropy coder

The last operation of the baseline JPEG algorithm is entropy coding. Its goal is to increase the compression performance of the encoder by taking advantage of the statistics from the data at the output of the quantizer.

The entropy coder of JPEG can be better explained by means of an example. Let us consider that the zigzag-reordered, 64-element array containing the quantized DCT coefficients, is: $A = \{a_0, a_1, 0, a_3, 0, 0, 0, a_7, 0, 0, a_{10}, 0, \ldots, 0\}$, for $a_i \neq 0$. The first operation consists in the DPCM applied to the DC coefficient. Supposing that the value of the DC coefficient of the previous 8x8 pixels block was b_0, then the DC value to be coded will be: $(a_0 - b_0)$. Runlength coding is applied to the AC coefficients; its result is a sequence of 2-element arrays $(r; a_i)$, where r represents the number of zeros preceding a non-zero a_i coefficient. Thus, after the DPCM and runlength coding, the sequence A, becomes: $B = \{(a_0 - b_0), (0; a_1), (1; a_3), (3; a_7), (2; a_{10}), EOB\}$, where EOB is a special code which indicates that the rest of the values, until the end of the sequence, are zeros.

For all the possible values that an a_i coefficient can take, different categories C_i are defined. Each category indicates a range to which the magnitude of a_i belongs. An extra number D_i is thus needed, in addition to C_i, to completely specify the value of a_i. Each pair of values $(r; C_i)$ is used to address a table that contains the Huffman codes. The splitting operation of each a_i into two numbers: C_i and D_i, is made in order to reduce the number of entries in the Huffman table.

The architecture of the entropy coder is shown in Figure 9. The zigzag reordering, required prior to the entropy coding, was implemented by a RAM with different read and write sequences. The DPCM and runlength operations are implemented by means of a subtractor and a zero detection counter, respectively, plus additional control logic and registers. The ROM contains the Huffman codes; since they are inherently of variable length (minimum 2 bits and maximum 16 bits, for the table that was chosen), a bit packer [17] has been added at the output of the circuit. The function of the bit packer is to concatenate consecutive codewords in order to output, at irregular intervals, words of fixed length.

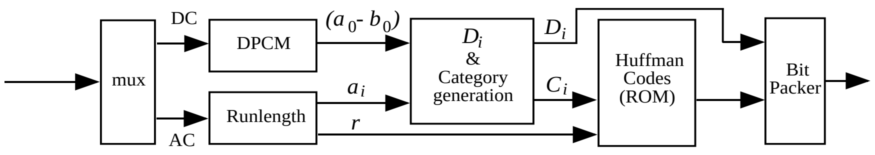


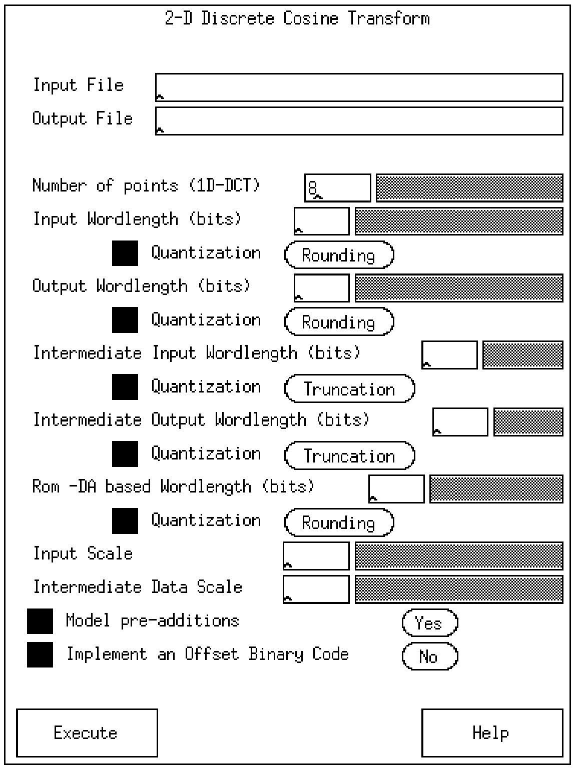
FIGURE 9. Architecture of the entropy coder.

5 Bit-true Level Modeling

The motivation for implementing a bit-true level model, is to assure the functionality of the architectures, the scheduling, the resource allocation and to find the optimum wordlength of the signals and coefficients of the algorithm.

Based on the programming framework of the high level implementation, a bit-true level model of the baseline JPEG algorithm was built. Each process of the algorithm was again implemented with C code and converted into a Khoros routine. However this time, the arithmetic operations are all carried out in binary arithmetic, modeling accurately the same processing and dataflow as would be executed by the corresponding processors and architectures described in section 4.

The GUI corresponding to the bit-true level model of the 2-D DCT is shown in Figure 10. Both rounding and truncation can be modeled. *Input*


2-D Discrete Cosine Transform

Input File

Output File

Number of points (1D-DCT) 8

Input Wordlength (bits)

Quantization Rounding

Output Wordlength (bits)

Quantization Rounding

Intermediate Input Wordlength (bits)

Quantization Truncation

Intermediate Output Wordlength (bits)

Quantization Truncation

Rom -DA based Wordlength (bits)

Quantization Rounding

Input Scale

Intermediate Data Scale

Model pre-additions Yes

Implement an Offset Binary Code No

FIGURE 10. GUI of the 2-D DCT bit-true level model.

Wordlength refers to the number of bits per pixel required at the input of the 2-D DCT circuit. The original number of bits of an original image (e.g., 8 bits for an image of 256 gray levels) might be increased by padding trailing or leading zeros, to comply with the control signals in bit-serial structures.

Output Wordlength refers to the number of bits required to represent the 2-D DCT coefficients. As pointed out in section 2, the 2-D DCT is computed by means of two successive 1-D DCTs. The *Intermediate* parameters in Figure 10 refer to the data between the two 1-D DCTs. The wordlength of the DAP's ROM refers to the number of bits required to represent the lookup table coefficients given by Equation (1.11). In Figure 7 this wordlength is denoted by W_c.

The *Scale* parameters are required to fix the binary point to the same position for all the data registers. In general the scale factors are chosen to be a positive integer power of 2, in order to facilitate the implementation by means of simple shifts.

The *Model pre-additions* parameter allows the user to decide whether the implementation of Equation (1.7) or (1.13) should be modeled. As the adder/subtractors in Figure 6 confirm, this option has been set to *Yes*, for our JPEG circuit.

The Offset Binary Coding (OBC) technique allows a binary table to be reduced from 2^N to 2^{N-1} elements, as explained in [18]. Though this option was not selected for our circuit (the reduction of a DAP's ROM size from 16 to 8 words was not very significant, due to the required additional logic and control), its model was implemented for experimental purposes.

For the 2-D DCT circuit, the optimum DAP's data- and ROM-wordlength found was 12 and 10 bits respectively. The intermediate input and output wordlength was evaluated to 12 bits. Simulations showed no difference in the results between using rounding or truncation. Thus, truncation mode was used, due to its simpler implementation.

The simulations of the bit-true level model of the quantizer circuit gave as a result a ROM wordlength value of 9 bits and an input/output data wordlength of 12 bits. No bit-true level model was developed for the entropy coder, due to the fact that the Huffman coder does not involve arithmetic operations.

6 Layout Design

By layout design we refer to the translation of the architectures into a CMOS circuit, which can be characterized by its speed, power consumption, etc.. Several software environments exist on the market today, to design circuits at any level (i.e., behavioral, logic, or transistor level). In our case, we have used the Compass environment [19], which contains a set of tools that aids the designer during the different stages of the circuit development.

When a module of an architecture is highly regular [21], the layout editor tool was used for developing full custom modules. The layout editor allows the design of circuits at the transistor level. For highly regular structures, this does not penalize the time for development since only a few cells have to be designed. On the other hand, minimum silicon area, minimum power consumption and high clock frequency circuits are achieved.

For less regular structures, the standard cell approach was applied. A cell library containing a collection of logic and arithmetic circuits (e.g., inverters, ANDs, NORs, latches, flip-flops, adders) are available to the user. By selection, arrangement and interconnection, the circuit designer is able to build circuits of higher complexity. The Logic Assistant tool of Compass is well suited to this task.

Cell compilers were also used during the datapath development of the JPEG circuit, mainly to generate the RAM for matrix transposition in Figure 6, and the ROM containing the Huffman tables [20] of the entropy coder in Figure 2.

At each hierarchical level of the design, extensive simulations were carried out with both the Mixed-Mode and SPICE simulators. This allows verification of the functional correctness of the circuit, and the extraction of characteristics such as the maximum attainable working frequency.

7 Results

The layout of the JPEG coder circuit is shown in Figure 11. The area of the chip is 4.6 x 3.1 mm^2 $\approx$ 14.5 mm^2. It was implemented in the 1.2μm 5V CMOS CMN12 process from VLSI Technology Inc. On the left part of the layout are located the SRBs, the quantizer and the two DAPs. All these circuits were built with full-custom layouts, a fact that is reflected by the very small area of the modules. The reutilization of the basic layout cells was efficiently maximized. The RAM for executing the matrix transposition and the zigzag reordering is located at the bottom-left corner of the circuit. The Huffman coder was realized by using the standard cell approach. It corresponds to the regular structure composed of rows of cells in the right part of the layout. The two blocks in the middle of the layout correspond to the ROMs containing the Huffman codes tables.

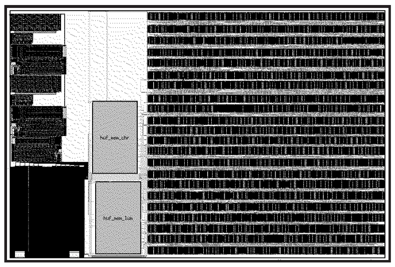


FIGURE 11. JPEG coder circuit.

At a clock frequency of 36 MHz, this circuit is able to process 25 CIF (Common Intermediate Format: 352 x 288 pixels) images per second. Thus, it is also suitable for motion JPEG (MJPEG) or for the non-recursive path of the H.261 [22] low-bit rate video coder. With a negligible increase in the silicon area, this circuit has been modified to feature four power management modes of operation [23]. Each mode of operation allows the user to trade image quality for power consumption; an option that can be very useful in portable multimedia equipment.

7.1 Extensions for Video Coding

Motion JPEG, though formally not supported by any coding standard, represents one practical solution for DCT-based video coding. That is, if an available JPEG encoder is fast enough, then one could use it to compress a video sequence by coding each frame as an independent still image. On the other hand, a more effective video coding system can be seen as a still coding scheme, to which a module to reduce the frame-to-frame redundancies has been added. Motion compensation is a very well-suited technique for reducing the interframe redundancies, related to the movements of objects across consecutive frames in a video sequence. For example, in the standards H.261, MPEG-1 [24] and MPEG-2 [25], the principle is to use a frame, plus the motion information, as a prediction to the immediately following frame(s). The prediction error frame—the current frame minus its prediction frame—is then coded as in a classic DPCM scheme.

The estimation of the motion vectors is the most computationally intensive operation in a video coder. In accordance with architectures that were discussed in section 4, a bit-serial VLSI architecture for a motion estimation algorithm was reported in [26]. In the frame of the methodology, the incorporation of the study of the interframe redundancy reduction modules has to be added, in order to evaluate the implementation of video coding systems.

8 Conclusions

A semi-custom methodology for the VLSI implementation of image compression systems was reported. The software environments were described along with an example of the implementation of a JPEG coder circuit. Though image compression was addressed in this chapter, the methodology could equally be applied to other image processing applications. The intermediate results of this methodology can also be used to develop solutions for other kinds of technologies (DSP, FPGA, etc.).

The main objective of the chapter was to give a general overview of a methodology, and corresponding CAD framework, that transforms an image processing algorithm into an integrated circuit. Readers unfamiliar with VLSI circuits are reassured that some of the descriptions and options, in the previous sections, were not described in detail, due to unavoidable space limitations. For further descriptions the reader is invited to consult the given references.

9 Acknowledgements

This work was supported by the Swiss National Science Foundation under Grant FN 2000-40'627.94, and by the Laboratory of Microtechnology (LMT EPFL). The latter is an entity common to the Swiss Federal Institute of Technology, Lausanne, and the Institute of Microtechnology, University of Neuchâtel.

The support provided by the Swiss Federal Office for Education and Science under Grant 874-01-49 is also acknowledged.

10 References

[1] R. C. Gonzalez and R. E. Woods, *"Digital Image Processing"*, Addison-Wesley, Reading, MA, USA, 1992.

[2] M. Rabbani and P.W. Jones, *"Digital Image Compression Techniques"*, Vol. TT 7, SPIE Optical Engineering Press, Bellingham, WA, USA, 1991.

[3] M. Kunt, G. Granlund and M. Kocher, *"Traitement Numérique des Images"*, Presses Polytechniques et Universitaires Romandes, Collection Electricité, Traitement de l'Information, Vol. 2, Lausanne, Switzerland, 1993. (In French).

[4] Philips, *TriMedia VLIW-Based PCI Multimedia Processor System (TriMedia White Paper)*, 1995.

[5] Texas Instruments, *TMS320C8x*, Technical Documentation, 1995.

[6] W.B. Pennebaker and J.L. Mitchell, *"JPEG Still Image Data Compression Standard"*, Van Nostrand Reinhold, New York, NY, USA, 1993.

[7] K. R. Rao, and P. Yip, *"Discrete Cosine Transform: Algorithms, Advantages, Applications"*, Academic Press, Boston, MA, USA, 1990.

[8] V. Bhaskaran and K. Konstantinides, *"Image and Video Compression Standards. Algorithms and Architectures"*, Kluwer Academic Publishers, Boston, MA, USA, 1995.

[9] J. Bracamonte, *"A high and bit-true level implementation of the baseline JPEG image compression algorithm"*, Internal Report IMT, Institute of Microtechnology, University of Neuchâtel, Switzerland, 1996.

[10] D. Rasure, D. Arguiro, T. Sauer and C. William, "A visual language and software development environment for image processing", *Int'l J. of Imaging Systems and Technology*, Vol. 2, 1990, pp. 183–199.

[11] ITU-T Recommendation T.81 *"Digital compression and coding of continuous-tone still images"*, September, 1992.

[12] Khoral Research, Inc., *Khoros Programmer's Manual*, 1995.

[13] R. I. Hartley and K. K. Parhi, *"Digit-Serial Computation"*, Kluwer Academic Publishers, Boston, MA, USA, 1995.

[14] P. Denyer and D. Renshaw, *"VLSI Signal Processing: A Bit-serial Approach"*, Addison-Wesley, VLSI System Series, 1985.

[15] A. Peled and B. Liu, "A new hardware realization of digital filters", *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. ASSP-22, No. 6, Dec. 1974, pp. 456–462.

[16] U. Sjöström, *"On the design and implementation of DSP algorithms: An approach using wave digital state-space filters and distributed arithmetic"*, Ph.D. Thesis, University of Neuchâtel, Switzerland, 1993.

[17] S-M. Lei and M-T. Sun, "An entropy coding system for digital HDTV applications", *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 1, No. 1, March, 1991, pp. 147–155.

[18] S. G. Smith and P. B. Denyer, *"Serial-Data Computation"*, Kluwer Academic Publisher, Boston, MA, USA, 1988.

[19] COMPASS Design Automation, Inc.: Manuals, COMPASS, San Jose, CA, USA, 1993.

[20] C. Henny, *"A VLSI implementation of a Huffman Coder"*, Diploma Project, University of Neuchâtel, Switzerland, August, 1995.

[21] N. Weste and K. Eshraghian. *"Principles of CMOS VLSI design: A Systems Perspective"*, Addison-Wesley, VLSI System Series, 2nd Edition, 1993.

[22] ITU-T Recommendation H.261 *"Video codec for audiovisual services at* $p \times 64$ *kbits"*, March, 1993.

[23] J. Bracamonte, M. Ansorge and F. Pellandini. "VLSI systems for image compression. A power-consumption/image-resolution trade-off approach", *Proc. Conf. on Digital Compression Technologies & Systems for Video Communications*, Berlin, Germany, Oct. 7-11, 1996.

[24] ISO/IEC JTC1 CD 11172, "Coding of moving pictures and associated audio for digital storage media up to 1.5 Mbits/s", International Organization for Standardization (ISO), 1992.

[25] ISO/IEC JTC1 CD 13818, "Generic coding of moving pictures and associated audio", International Organization for Standardization (ISO), 1994.

[26] J. Bracamonte, I. Defilippis, M. Ansorge and F. Pellandini. "Bit-serial parallel processing VLSI architecture for a block matching motion estimation algorithm", *Proc. of the International Picture Coding Symposium PCS'94*, Sacramento, CA, USA, Sept. 21-23, 1994, pp. 22–25.